طريق البرمجة في



هاني الأتاسي – الفريق العربي للبرمجة

آخر تعديل في 11/20/2001

ملاحظة : هذا الكتاب هو قيد الاعداد حاليا وهو لم يكمل .

عن المؤلف :

هاني الأتاسـي (atassi@arabteam2000.com)

الفهرس

مقدمة

أساسيات البرمجة

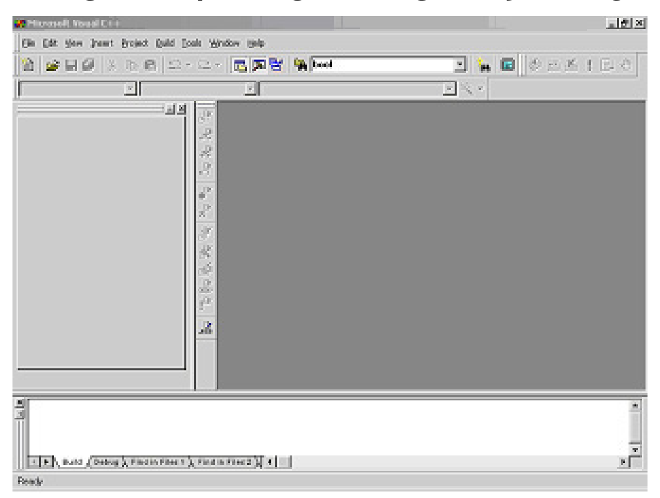
في هذا الدرس سوف أشرح كيفية كتابة أول برنامج لك بالسي++ باستخدام Microsoft Visual C++ 6.0 ، وهي البيئة التي سوف نتعامل معها من هذه اللحظة حتى الانتهاء من الدروس . هذا لا يعني أنه لايمكنك استخدام مترجمات أخرى مثل Borland C++ ولكن سوف يكون شرحي كله على Visual C++ وذلك من أجل التمكن من استخدام هذه البيئة في حال أردنا للانتقال إلى برمجة تطبيقات وندوز باستخدامها .

هذا الدرس سوف أشرح فيه أساسيات ومفاهيم عامة ولكن الغرض الأساسي منه هو فقط من أجل وضع رجلنا على أول الطريق وبداية المشوار .

1.1 البرنامج الأول

في هذا البرنامج ببساطة سوف نقوم بطباعة الجملة المشهورة "Hello World" على الشاشة . حسنا لماذا لا نجهز الشواء ونقوم بتشغيل الفيجوال سي الآن !

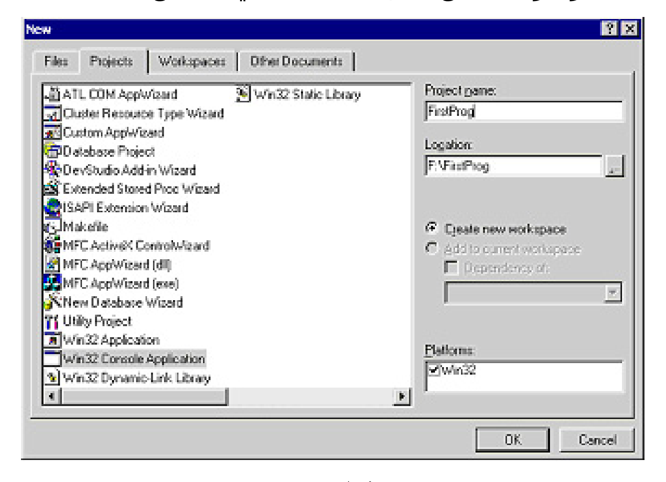
إذا تمت عملية التشغيل بنجاح طبعا سوف تحصل على شكل مشابه إلى الشكل 1.1 .



شكل 1 -1

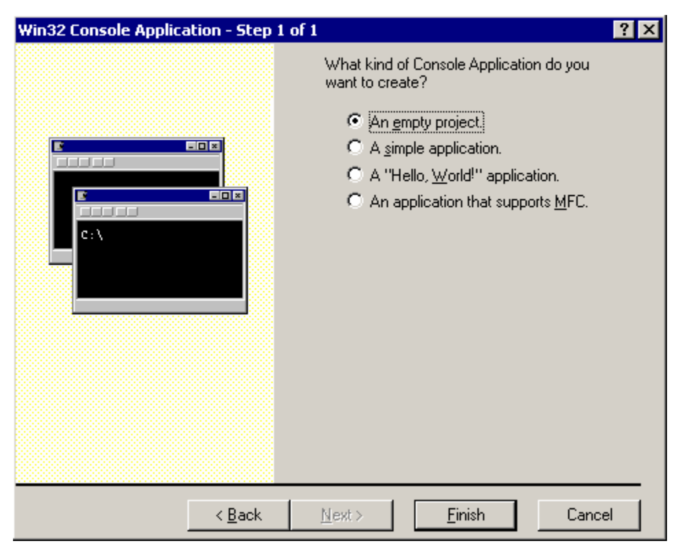
الخطوات اللازمة من أجل البدأ في أي مشروع في هذه الدروس هي التالية :

- اختر New من القائمة File وسوف تحصل على نافذة كما في الشكل 2-1 .



شكل 1 -2

- من الصفحة Projects قم باختيار Win32 Console Application . ومن ثم اكتب اسم المشروع في الخانة Project Name ، وليكن على سبيل المثال اسم المشروع FirstProg .
- بشكل افتراضي سوف يقوم الفيجوال سي++ بانشاء دليل بنفس اسم المشروع ويضع فيه جميع الملفات التي تضيفها لاحقا للمشروع .
- اضغط على OK . وحينها سوف تحصل على النافذة كما في الشكل 1-3 .

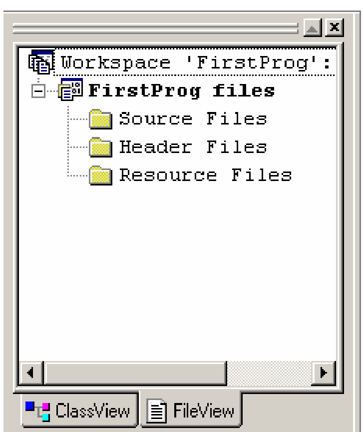


شكل 1 -3

- تأكد من اختيار An empty project ومن ثم اضغط على Finish .

من خلال الخطوات السابقة يكون قد انشأت مشروع برمجي وما يتبقى هو اضافة ملفات السي للمشروع . يمكنك أن تتخيل المشروع كبيئة عمل منظمة لبرنامجك ، حيث تقوم باضافة أي عدد من الملفات للمشروع وتنظيمها تماما كما تنظم الملفات في مستكشف الوندوز .

طبعا بما أننا اخترنا مشروع فارغ فهذا يعني أن المشروع الجديد الذي بنينا فارغ ولا يحتوي على أي من الملفات. عند عودتك إلى البيئة الرئيسية للفيجوال سي سوف تلاحظ أن القسم الأيسر صار يحتوي على اسم مشروعك الذي كتبته وهي مشابهة للشكل 1-4 .



شكل 1 -4

الفيجوال سي قام بوضع ثلاث مجلدات لنا داخل المشروع من أجل وضع ملفاتنا . هذه المجلدات هي وهمية أي لا يوجد لها مقابل على القرص الصلب . الأول وهو Source Files وتضع فيه جميع ملفات المصدر لمشروعك أي الملفات التي تحتوي على الكود ، الثاني Header Files وتضع فيه جميع ملفات التعاريف أو ال Header File كما سوف ترى لاحقا ، أما الثالث Resource Files وهو لايهمنا في هذه الدروس لأنه مختص للبرمجة لبيئة الوندوز.

الآن من أجل كتابة البرنامج الأول يجب علينا من انشاء ملف كود جديد وهذا من الأمر New من القائمة File . سوف تحصل على نفس النافذة في الشكل 1-1 ولكن هنا الصفحة Files هي الظاهرة عوضا عن Projects . طبعا يمكنك أن تخمن ماهو الذي سوف نختاره . طبعا هو C++ Source File . بعد أن تختاره اكتب اسم الملف وليكن test ومن ثم اضغط موافق .

سوف تجد أنه تم اضافة الملف test.cpp إلى الدليل الوهمي Source Files وذلك بسبب أن هذا الملف سوف يحتوي على كود للمشروع . الآن قم بكتابة الكود التالي في الملف الفارغ الذي أنشئته .

```
#include <iostream.h>

int main()
{
     cout << "My first program.";
     return (0);
}
```

PROGRAM 1

يجب أن تنتبه أن لغة السي++ تفرق بين الأحرف الكبيرة والأحرف الصغيرة . على سبيل المثال فإن اسم التابع main يجب أن يكتب كما هو ولا يمكن كتابته ك Main أو MAIN وهي كلها تعتبر كأسماء مختلفة عن التابع main .

1.2 ترجمة وبناء المشروع

من أجل تنفيذ البرنامج السابق وبناء ملف تنفيذي exe قابل للتنفيذ في أي وقت يجب أولا ترجمته وتحويله من اللغة المقروئة إلى لغة الآلة التي كلها أصفار وواحدات وبعدها يمكن تنفيذه .

عملية ترجمة المشروع وبناء الملف التنفيذي في الفيجوال سي++ تتم بخطوة واحدة وهي عن طريق اختيار الأمر Build FirstProg.exe من القائمة Build أو الضغط على F7 . ومن أجل تنفيذ المشروع أو الملف التنفيذي اختر الأمر Execute FirstProg.exe من القائمة Build أو عن طريق المفتاحين Ctrl+F5 . طبعا يمكنك أولا بناء الملف التنفيذي ومن بعدها تنفيذه بخطوتين ولكن الأمر Execute يقوم بشكل آلي بعملية الترجمة وبناء الملف التنفيذي إذا دعا الأمر إلى ذلك .

إذا كان كل شئ على مايرام سوف ترى نافذة مماثلة لنافذة الدوس أو ال Command Line وسوف ترى العبارة My first program في أعلى الشاشة كما في الشكل 1-4 .

```
"F:\FirstProg\Debug\FirstProg.exe"
My first program.
Press any key to continue
```

شكل 1 -5

1.3 الأهم من تعلم لغة البرمجة

هل سئلنا أنفسنا في لحظة من اللحظات ماهم الأهم من تعلم لغة البرمجة من أجل أن نصبح مبرمجين ؟ بصراحة الإجابة على هذا السؤال ضخمة جدا ويوجد كتب كاملة عن هذا الموضوع أو عن موضوع هندسة البرمجيات . الإجابة هي كلمتين طبعا وهي هندسة البرنامج ، تماما كما يفعل إخواننا في الهندسة المدنية أم المعمارية . تخيل برنامجك هو عبارة عن عمارة أو بناء تريد تشييده ، سوف تبدأ بالتصميم وتحليل مواردك وامكانياتك ومن ثم وضع (سكيتش) أو تصميم أولي للبناء وبعد ذلك تبدأ بالبناء بلوك بلوك وطابق طابق .

لن أتطرق في هذه الدروس إلى مواضيع التصميم وتحليل الموارد لأنها قد تكون خارجة عن اطار هذه الدروس ولكن كدروس لتعليم لغة السي++ يجب أن أنوه إلى مواضيع تنظيم البرنامج والتي هي جزء لا يتجزء من هندسة البرنامج.

سوف أتطرق الآن إلى أهم الأمور وأسهلها وهي اضافة التعليقات في برنامجك . التعليقات هي جزء من البرنامج الذي يهمل أثناء عملية ترجمة البرنامج وهي تفيد في تفسير الخطوات التي تقوم بها . سوف أطلب منك مقارنة البرنامجين التالين وملاحظة أيهما أسهل للفهم بغض النظر عن لغة السي++ المستخدمة التي قد لا تكون ملم بها في هذه اللحظة .

PROGRAM 1.2 (a)

```
int main()
{
int sum;
sum = 10 + 20;
cout << sum << endl;
return (0);
}
```

PROGRAM 1.2 (b)

```
/*
**   This program calculates the sum of 10 and 20 and prints it on screen
**   Programmed by : Anonymous
**   Date : 1/1/3000
*/

int main()
{
     int sum;                 // This will hold the sum of the numbers

     sum = 10 + 20;           // Store 10 + 20 in sum
     cout << sum << endl;     // Print the sum value

     return (0);              // End the program
}
```

طبعا سوف تقول أن البرنامج الثاني أسهل للقراءة والفهم من الأول لعدة أسباب .

- استخدام التعليقات أينما دعت الحاجة لها .
- وضع أسطر فارغة بين قسم وآخر في الكود ، الأقسام تكون مترابطة منطقيا فيما بينها .
- ترك مسافات بادئة من أجل سهولة القراءة وتتبع البرنامج .

بالنسبة إلى التعليقات فيمكن كتابتها بشكلين كما هو موضح في البرنامج الأخير . الأولى باستخدام // وهي تقوم باعتبار كل الذي بعدها حتى نهاية السطر كتعليق . أما الشكل الآخر باستخدام */ و /* الرمز الأول يفتح فقرة تعليق والرمز الثاني يغلق هذه الفقرة ، ذكرت فقرة لأن هذا النوع من التعليق ممكن أن يمتد على عدة أسطر أو يمكن أن يكون في سطر واحد .

1.4 ملخص الدرس

- من أجل البدأ بمشروع برمجي يجب انشاء مشروع جديد في الفيجوال سي++ عن طريق اختيار New من القائمة File .
- يتم انشاء ملفات كود جديدة واضافتها للمشروع أيضا من خلال الأمر New من القائمة File .
- قمنا بكتابة أول برنامج في السي++ وتنفيذه .
- تعلمنا كيفية كتابة تعليقات في البرنامج .

من أجل تنزيل ملفات مشروع هذا الدرس انقر هنا .

الأنواع والعوامل والتعابير – Types, Operators, and Expressions

البرنامج في السي++ يصف العمليات والمهام التي سوف تطبق على العناصر التي تحمل البيانات . من الأمور المهمة جدا كما هو الحال في باقي اللغات هو عملية تحديد ما هو نوع هذه العناصر أو بكلمة أخرى أنواع المعطيات. يمكن تصنيف هذه الأنواع في السي++ كأنواع أساسية وأنواع مشتقة . على سبيل المثال الأنواع المشتقة هي تلك الأنواع التي تعتمد على الأنواع الأساسية مثل المصفوفات .

إن البيانات التي يتم التعامل معها في البرنامج يمكن تصنيفها إلى قسمين . الأولى هي التي تبقى ثابتة طول فترة تنفيذ البرنامج ، أما الثانية هي التي تتغير قيمتها . الاسم الذي يطلق على الأولى هي الثوابت أو Constants أما الثانية فسوف نطلق عليها اسم المتحولات أو Variables . طبعا كلا الصنفين السابقين يجب أن تنطبق عليه ماذكرناه بالنسبة إلى الأنواع ؛ على سبيل المثال المتحولات يجب تحديد نوعها حتى نستطيع التعامل معها .

أما التعبير expression فهو عبارة عن الخطوات والقوانين التي سوف يتم من خلالها حساب قيمة معينة . التعبير يتكون من مجموعة من الحدود operands أو القيم و العوامل operators . لغة السي++ غنية جدا بالعوامل التي تعكس العمليات التي تتم في المعالج مثل عمليات الجمع والضرب والازاحة . العوامل يمكن تصنيفها إلى عدة أصناف أيضا فمنها عوامل من أجل العمليات الحسابية كالجمع والضرب ومنها عوامل المقارنة كالأصغر والأكبر . سوف نقوم بشرح بعض العوامل في هذا الدرس مع تكميلها في باقي الدروس .

- هذا الدرس يوجد فيه عدد هائل من المعلومات مع قليل من التطبيق ، لذا ليس من الضروري فهمه بشكل كامل ولكن يجب قرائته بشكل كامل والرجوع إليه إذا دعت الحاجة في الدروس اللاحقة .

2.1 الثوابت الصحيحة – Integer Constants

هذه الثوابت هي عبارة عن أعداد قد تكون موجبة أو سالبة . ويمكن تمثيلها كأعداد بالنظام العشري أو الثماني أو الست عشري . الأعداد بالنظام العشري تتكون من الأرقام من 0 إلى 9 والرقم الأول يجب أن يكون غير الرقم 0 . وكأمثلة على هذا لدينا الأعداد :

1234 801 9999 10

كلنا نعلم أن الحاسب الآلي يستطيع التعامل مع مجال محدد للأعداد . دقة هذه الأعداد تعتمد على المساحة التي يحجزها المترجم للعدد ، وهذا يختلف من حاسب إلى آخر ففي الأجهزة التي تحتوي على مسجلات بحجم 16بت فإن ال integer سوف يحجز له 16بت وبالتالي فإن مجاله يتراوح بين 32768- إلى 32767 . في الأجهزة الحالية وتحت نظام وندوز 32بت فإن ال integer يحجز له 32بت وبالتالي يكون مجاله بين 2147483648- إلى 2147483647 .

بالنسبة إلى الأعداد في النظام الثماني فإنها يجب أن تبدأ بالرقم 0 وهذه أمثلة عليها :

0235 0123 0777 020

أما الأعداد بالنظام الست عشري فهي تبدأ بالرمز 0x أو 0X وهذا أمثلة على هذا :

0x0A2F 0XECA2 0x1234 0xFFFF

في حال تخطى الثابت المجال المسموح فيه فإن معظم المترجمات تقوم بالتحذير عن ذلك .

- في الأجهزة 16بت فإنه يمكن اجبار المترجم على اعتبار العدد ذو حجم 32بت باضافة اللاحقة L أو (الحرق الصغير l) مثل التالي 0x23A1L أو 982L .

2.2 الثوابت للأعداد الحقيقة – Floating point Constants

أي رقم يحتوي على قسم كسري يعتبر عدد حقيقي . وتكتب هذه الأرقام بصيغتين . الصيغة المباشرة والبسيطة أمثلة على هذا :

3.1415926 12.0 0.1 0.00001

النقطة التي تفصل القسم الحقيقي والكسري لا تشترط أن تحطاط برقمين حيث يمكن كتابة الرقم الثانية والثالث في المثال السابق كالتالي :

12. .1

ولكن يفضل ، على كل الأحول ، استخدام الصيغة التي تجعل البرنامج قابل للقراءة بشكل واضح .

الصيغة الثانية أو مايسمى بالتمثيل العلمي وهو عبارة عن رقم مثل السابق ولكن مضروب برقم مرفوع إلى العدد 10 هذا الأس يمثله الحرف e أو E أمثلة على هذه الصيغة هي :

1.0e2 (= 100)   12.34e-3 (= 0.01234)

2.3 ثوابت المحارف – Character constants

الثابت المحرفي هو أي حرف كتب بين اثنتين من الفاصلة العلوية الواحدة أو single quotes ؛ مثل 'X' . إن قيمة الثابت المحرفي هي عبارة عن قيمة هذا الحرف في جدول المحارف في النظام أو الجهاز . من أشهر جداول المحارف استخداما هو ASCII (American Standard Code for Information Interchange) . ففي جدول ASCII تبلغ قيمة المحرف 'X' القيمة 88 .

يوجد أيضا العديد من المحارف في لغة السي والتي تسمى بمحارف الهروب أو escape character . وهي تستخدم للتعامل مع المحارف التي من الصعب التعامل معها أو من الصعب ادخالها من لوحة المفاتيح مباشرة . وهي تبدأ بالرمز \ ثم يليه محرف واحد . المحرفين السابقين يعتبرو كمحرف واحد من وجهة نظر المترجم ، سوف أسرد هذه المحارف هنا :

Alarm bell	\a
New line	\n
Horizontal tab	\t
Vertical tab	\v
backspace	\b
Carriage return	\r
Formfeed	\f
Single quote	\'
Double quote	\"
backslash	\\
Question mark	\?
NULL character	\0

الآن على سبيل المثال إذا أردت التعامل مع المحرف \ فيجب أن تكتب '\\' عوضا عن '\' . أيضا بالإضافة إلى محارف الهروب السابقة يمكن أن تتكون سلسلة الهروب من المحرف \ يتبعها x ومن ثم واحد أو أكثر من الأرقام الست العشرية . الناتج هو عبارة عن رقم ست عشري يمثل دليل في جدول المحارف في الجهاز على سبيل المثال فإن المحرف 'X' قيمته 0x58 في جدول ASCII وبالتالي يمكن تمثيله أيضا بالصيغة '\x58' . حتى نكون دقيقين أكثر الصيغة الأخيرة تستخدم لطباعة المحارف الموجودة في جدول ASCII التي لا يمكن ادخالها من لوحة المفاتيح كالمحارف اللاتينية وغيرها .

المحرف الأخير الموجودة في القائمة السابقة يعتبر المحرف الأول في جدول ASCII ويسمى بمحرف الصفر أو ال NULL Character '\0' . هذا المحرف له استخدامات خاصة في السي وهذا ماسوف نراه لاحقا في هذا الدرس .

2.4 ثوابت السلاسل النصية – String constants

السلسلة النصية هي مجموعة من المحارف محاطة بزوج من الفاصلتين العلويات أي " . ويمكن أيضا استخدام نفس محارف الهروب السابقة بداخل السلسلة النصية . وكأمثلة على السلاسل النصية هي :

```
"Hello world"
"The C++ Programming Language"
""
"He said \"Good morning\""
"This string terminates with a newline \n"
```

أيضا السلسلة النصية يمكن أن تمتد إلى أكثر من سطر عن طريق استخدام المحرف \ في آخر السطر الأول ومنه الاستمرار في السطر الثاني وكمثال على هذا إليك التالي :

"This string extends \
over two lines."

المترجم يعتبر السلسلة السابقة كسلسلة واحدة أي يقوم بتجاهل المحرف \ بالاضافة إلى رمز السطر الجديد الذي بعده . وهي مشابهة تماما إلى السلاسلة :

"This string extends over two lines."

وأيضا السلسلتين المنفصلتين التي تلي أحدهما الأخرى يقوم المترجم باضافتهما إلى بعضهما لتكوين سلسلة واحدة . مثلا هذا المثال :

"This " "string"

يحول إلى :

"This string"

الطريقة السابقة جدا مهمة في قص السلاسل الطويلة ووضعهم على أكثر من سطر.

أيضا الأمر الهام جدا بالنسبة إلى موضوع السلاسل في السي++ فإن أي سلسلة سوف تنتهي ب NULL Character أو '\0' . وهذا المحرف هو الذي يحدد انتهاء السلسلة النصية . هذا يعني إذا كانت السلسلة تحتوي على عدد N من المحارق فإنها حقيقة سوف تحتوي على N+1 من أجل احتواء محرف النهاية . لذلك تسمى السلاسل النصية في لغة السي ب Null terminated string أو السلسلة المنتهية بمحرف الصفر . وإليك هذا الشكل :

h	e	l	l	o	\0

شكل 2 -6

بالاعتماد على سبق يجب أن نلاحظ أن المحرف 'X' والسلسلة "X" ليسا نفس الشئ فالمحرف في معظم الأجهزة عبارة عن 8بت أو بايت أما السلسلة "X" فسوف يحجز لها محرفين وبالتالي سوف تكون بايتين . وبالتالي يجب الانتباه في حالة استخدام المحارق والسلاسل والتفريق فيما بينهما .

2.5 المعرفات – Identifiers

لغة السي++ مثل باقي اللغات تتطلب منك اعطاء أسماء معينة للمعطيات التي تستخدمها في برنامجك . هذه الأسماء تدعى بالمعرفات أو Identifiers ويتم وضعها أو تكوينها من قبل المبرمج . ويجب أن يصنع المعرف بالاعتماد على القاعدة التالية :

المعرف هو عبارة عن خليط من المحارف والأرقام والتي يجب أن تبدأ بمحرف . الرمز (_) أو Underscore يمكن استخدامه بالمعرف وهو يعتبر كحرف عادي .

من القاعدة السابقة يمكن أن نضع بعض الأمثلة لمعرفات تعتبر صحيحة لمترجم السي++ :

time	counter	BUFFER
x	unit_cost	h2o
_MAX	programName	AVeryLongIdentifier

يعتبر أي معرفين هما واحد في حالة كان لهما نفس الاملاء ونفس حالة الأحرف (الصغيرة أم الكبيرة) . وبالتالي المعرفين abcd و abcD لا يعتبرا واحد.

يفضل أن يعطي المبرمج اسماء للمعرفات تدل على استخدامها . على سبيل المثال إذا كان المبرمج يتعامل مع الوقت في برنامجه فيفضل أن يسمي معرفاته ب hours و minutes و seconds وهذا طبعا أفضل بكثير من تسميتها h و m و s .

بعض المعرفات تكون محجوزة لاستخدام لغة السي++ وهي ماتسمى ب language keywords . السرد الكامل لهذه المعرفات معطى في الأسفل ولكن شرحها سوف ياتي لاحقا كل في قسمه .

asm	auto	bad_cast	bad_typeid
bool	break	case	catch
char	class	const	const_cast
continue	default	delete	do
double	dynamic_cast	else	enum
except	explicit	extern	false
finally	float	for	friend
goto	if	inline	int
long	mutable	namespace	new
operator	private	protected	public
register	reinterpret_cast	return	short
signed	sizeof	static	static_cast
struct	switch	template	this
throw	true	try	type_info
typedef	typeid	typename	union
unsigned	using	virtual	void
volatile	while		

بقي أمر هام يجب التنويه له وذلك أنه لا ينصح باستخدام رمـزين (_) فـي بدايـة المعـرف ، مثـل __MAX ، وذلـك بسبب أن المعرفات بهذا الشكل محجوزة لاستخدام المكتبات القياسية للسـي++ . لذلك تجنب من استخدامها .

2.6 تعريف المتحولات – Variable definition

من اسمها فإن المتحولات هي عبارة عن عناصر معطيات تتغير قيمها أثناء وقـت تنفيـذ البرنـامج . هـذه التحـولات يجب أن يكون لها نوع يتم تحديده حتى يستطيع المترجم التعامل معها وتوليد الكود الصـحيح . طبعـا إن عمليـة تحديـد نوع المتحول تعرف بتعريف المتحولات أو Variable definition . إن التعريف يكون على الهيئة التالية :

```
type-specifier    list-of-variables;
```

إن list-of-variables هي عبارة عـن معرفـات يفصـل بينهـا علامـة الفاصـلة هـذه المعرفـات تمثـل المتحـولات فـي برنامجك . كل متحول أو مجموعة من المتحولات يتم تحديد نوعها عن طريق ال type-specifier . إن الأنـواع الأسـاسـيـة في السـي++ يمكن اعتبارها :

```
int       (integer)
char      (character)
float     (single precision floating point)
double    (double precision floating point)
```

وكمثال على تعريف المتحولات :

```
int       hours, minutes, seconds;
```

هنا المتحولات hours و minutes و seconds كلهـا عبـارة عـن متحـولات لأعـداد صـحيحة integer وبالتـالي يمكـن تخزين أي عدد integer ضمنها واستخدام هذا العدد لاحقا في برنامجك .

طبعا يمكن أن يتم تعريف مجموعة من المتحولات ذات أنواع مختلفة وهذا طبعا يتم باستخدام الفاصلـة المنقوطـة كفاصل . ويفضل كتابة كل تعريف بسطر منفصل من أجل زيادة الوضوح . وهذا مثال على ذلك :

```
int       day, month, year;
float     centigrade, fahrenheit;
char      initial;
double    epsilon;
```

وعند التعريف يفضل تعريف المتحولات ذات العلاقة فيما بينها في مجموعة والباقي في مجموعة أخرى كالتالي :

```
int       counter, value;
int       day, month, year;
```

وأيضا يمكن كتابة التعريف على عدة أسطر كالمثال التالي :

```
int     day, month, year,
        hours, minutes, seconds;
```

أيضا يمكن اعطاء قيم معينة للمتحولات أثناء تعريفها وهذا يتم عن طريق استخدام رمز الالحاق (=) كالتالي :

```
int     sum = 0;
float   pi = 3.14;
```

المتحولات السابقة تبقى محتفظة بقيمتها حتى يتم تغييرها في البرنامج . أما المتحولات التي يتم تعريفها من غير اعطائها قيمة معينة فقيمتها غير معروفة ولا ينصح باستخدامها حتى يتم اعطائها قيمة معينة في البرنامج .

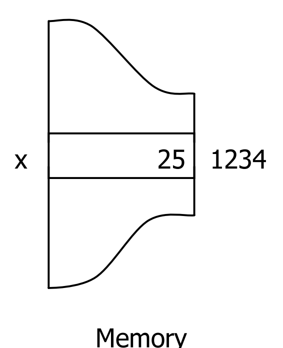
بصراحة فإن هذا العرض البسيط لتعريف المتحولات غير كافي وليس من المعقول اعطاء كل النواحي في تعريف المتحولات في الدروس الأولى لذلك سوف نتوسع في ذلك لاحقا . يكفي الآن أن تعرف أن كل متحول له الخواص التالية :

- إنها من نوع ما .
- تملك عنوان معين في الذاكرة .

كل متحول يجب أن يكون متواجد في الذاكرة في عنوان معين . ويقوم المترجم بالرجوع إلى هذا العنوان في حال وجد المتحول في السياق . على سبيل المثال اعتبر التعريف التالي :

```
int     x = 25;
```

على مستوى فهمنا فإنه يمكن القول أن تم حجز مساحة في الذاكرة من أجل استيعاب أرقام integer وهذه المساحة يتم التعامل معها عن طريق المعرف x . بالنسبة إلى المستوى المنخفض للحاسب فإن هذه المساحة تملك العنوان (على سبيل المثال 1234) . انظر الشكل 2-2 .



شكل 2 -7

2.7 الواصفات – Qualifiers

الواصفات في لغة السـي++ التي سـوف نشـرحها في هذه الفقرة هي :

- long و short .
- signed و unsigned .
- const .

الواصفين long و short يسـتخدان قبل النوع int من أجل التعديل فـي طـول المتحـول . إن اسـتخدام long يعطـي المتحول أكبر مجال ممكن للأعداد التي يمكن استخدامها . وكمثال على استخدام long :

```
long int        memory_address;
```

في معظم الأجهزة القديمة كانت ال int تأخذ مامقداره 16بت وال long مامقداره 32بـت . أمـا حاليـا تحـت الونـدوز وباسـتخدام الأجهزة الحديثة فإن ال int وال long كلاهما تأخذ مسـاحة مقـدارها 32بـت . أي لا يوجـد أي اخـتلاف بـين اسـتخدام int أو long . ولكـن مـن أجـل التوافقيـة بـين المترجمـات المختلفـة يجـب أن تعلـم أن int تتعلـق بالحاسـب المستخدم والمترجم المستخدم أما long فهي دوما تحجز أكبر مساحة ممكنة من أجل التعامل مع الأعداد وقـد تكـون 64بت في على معالجات 64بت الجديدة .

أما الواصف short الذي يوضع قبل int . فيدل على أن المتحول سـوف يسـتخدم مـن أجـل مجـال صـغير للأعـداد . وعلى الأغلب فإن short تقوم بحجز 16بت للمتحول . ويمكن اسـتخدامها في حالة أردنا التقليل من اسـتهلاك الذاكرة . ومثال على تعريف المتحولا باسـتخدام الواصف short :

```
short int       day_of_week;
```

في المثالين السـابقين يمكن تجاهل النوع int ، أي يمكن التعريف كالتالي :

```
long    memory_address;
short   day_of_week;
```

أيضا اسـتخدام long قبل double يعطي المتحول مجال أكبر من المجال المستخدم في double .

إن جميع ماذكرناه من متحولات كانت متحولات مؤشرة أي يمكن أن تحمل قيم موجبة وسالبة . أما في حالة أردنا أن يكون مجال الأعداد المستخدمة هي فقط موجبة فيجب أن نسـبق int أو char بالواصـف unsigned . هـذا يعنـي أن المتحول في هذه الحالة يعتبر موجب فقط ضمن المجال 0 إلى 65535 في حالة int ذات 16بت و من 0 إلـى 255 فـي حالة char . وكمثال على استخدام هذا الواصف :

```
unsigned int    natural;
unsigned        record_number;
unsigned char   i_am_byte;
```

إن اسـتخدام int في المثال السـابق كان اختيـاري ويمكـن تجاهلـها . والواصـف signed تعنـي أن المتحـولات هـي مؤشرة ولكن كما ذكرنا فإنه بشـكل افتراضي المتحولات حين حجزها تكون مؤشرة فإن اسـتخدام signed هـو تحصـيل حاصل .

أخيرا ، إن الواصف const يمكن استخدامه مع تعريف أي متحول وذلك لتحديد أن قيمة هذا المتحول لن تتغيـر أبـدا أثناء تنفيذ البرنامج . مثل هذه المتحـولات لا يمكـن اسـتخدامها علـى الطـرف الأيسـر لعلامـة الالحـاق (=) أو لا يمكـن استخدامها في أي شـكل من الأشـكال يـؤدي إلـى تغييـر قيمتهـا . المتـرجم فـي هـذه الحالـة يسـتطيع أن يضـع هـذا المتحول في ذاكرة قابلة للقراءة فقط أو يقوم بالتعديل على البرنامج بالشـكل الـذي يريـد (Optimization) . طبعـا فـي حالة اسـتخدمنا الواصف const يجب علينا اعطاء قيمة للمتحول وقت تعريفه كالتالي :

```
const double    pi = 3.1415926;
const int       student_number = 10;
```

من غير أي عملية تهيئة فإن هذا يعتبر خطأ أن تكتب :

```
const double    pi;
```

هنا لم نعطي المتحول pi أي قيمة وبما أنه لا يمكن اعطاء pi أي قيمة أثناء تنفيذ البرنامج فإن pi تبقى ذات قيمـة غير معرفة وبالتالي فإن مثل هذا النوع من التعريفات يعطينا خطأ أثناء الترجمة .

بما أنه انتهينا من ذكر أنواع المتحـولات وجميـع الواصـفات فإن الجـدول التـالي (جـدول 2-1) يـذكر جميـع الأنـواع الأساسية التي يمكن استخدامها في السـي++ (وأخص بالذكر فيجوال سـي++) . (الجدول مأخوذ من MSDN) .

Type Name	Bytes	Other Names	Range of Values
int	*	signed, signed int	System dependent
unsigned int	*	unsigned	System dependent
__int8	1	char, signed char	–128 to 127
__int16	2	short, short int, signed short int	–32,768 to 32,767
__int32	4	signed, signed int	–2,147,483,648 to 2,147,483,647
__int64	8	none	–9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
char	1	signed char	–128 to 127
unsigned char	1	none	0 to 255
short	2	short int, signed short int	–32,768 to 32,767
unsigned short	2	unsigned short int	0 to 65,535
long	4	long int, signed long int	–2,147,483,648 to 2,147,483,647
unsigned long	4	unsigned long int	0 to 4,294,967,295
float	4	none	3.4E +/- 38 (7 digits)
double	8	none	1.7E +/- 308 (15 digits)
long double	10	none	1.2E +/- 4932 (19 digits)

جدول 2 -1

- في أنظمة وندوز 95 ومابعدها (أي الأنظمة المعتمدة على 32بت) فإن int سوف تكون 4 بايتات أي أنها مثل ال long تماما . ولاحظ في الجدول السابق فإن الأنواع المسبوقة ب علامتين (_) هي اضافة من الفيجوال سـي++ أي أن استخدامها يعني أن برنامجك سوف يترجم فقط في بيئة الفيجوال سـي++ . لذلك لا ينصح باستخدامها بكثرة وبالتالي يفقد برنامجك خاصية التوافق مع باقي المترجمات . الشئ الوحيد الذي قد يفيدك هو __int64 التي تعطيك مجال ضخم جدا للاعداد ولكن يجب أن تنتبه على أن استخدام __int64 هو أبطأ من استخدام long أو int وخاصة في حساب التعابير الرياضية .

2.8 التعابير الحسابية – Arithmetic expressions

إن لغة السـي++ تدعم العمليات الحسابية العادية مثل الجمع (+) و الطرح (-) و الضـرب (*) والقسـمة (/) وأيضا تدعم عملية باقي القسمة للأعداد الصحيحة وهي (%) . العمليات السابقة كلها تطبـق علـى حـدين أي operands . المعاملات السابقة تسمى ثنائية أو binary لأنها تطبق على حدين . يوجد معاملات أخرى تطبق على حد واحد وهـي في هذه الحالة معامل الاشارة السـالبة (-) ومعامـل الاشـارة الموجبـة (+) ، الأخيرتـان تسـمى معـاملات أحاديـة أو unary وذلك بسبب أنها تطبق على حد واحد . وكأمثلة على هذا :

```
12 * length
-1024
3.14 * radius * radius
distance / time
money % 100
```

كما تلاحظ في المثال الأول فإن القيمة 12 ضربت مع ما يحتوي عليه المتحول length . أما في المثال الثاني فـإن معامل الاشارة السـالبة قام بعكس قيمة العدد 1024 الموجبة إلى سـالبة .

إن التعبير الحسابي في لغة سـي++ يتم حسابه على حسب أولوية العوامل المستخدمة في التعبير . إن أولوية العوامل هي التي تحدد ترتيب حساب العمليات الرياضية في التعبيـر . الجـدول 2-2 يوضـح المعـاملا الحسـابية حسـب أولويتها . طبعا الجدول الكامل للأولويات سوف يأتي لاحقا في دروس قادمة .

Operator	Description	Associativity
+	Unary plus	Right to left
-	Unary minus	Right to left
*	Multiplication	Left to right
/	Division	Left to right
%	Modules	Left to right
+	Addition	Left to right
-	Subtraction	Left to right

جدول 2 -2

من الجدول السابق تلاحظ أن معاملات الاشـارة لهـا أعلـى أوليـة ومعـاملات الجمـع والطـرح لهـا أدنـى الأولويـات . والضرب والقسمة وباقي القسـمة هـي فـي المنتصـف . وبالتـالي فـإن أي تعبيـر يحتـوي علـى خلـيط مـن المعـاملات السابقة فإن الذي ينفذ أولا هـو جميـع معـاملات الاشـارة إن وجـدت وبعـدها علـى الترتيـب حسـب الجـدول السـابق . وكمثال على هذا إليك بالتعبير التالي :

```
2 + 3 * 4
```

التعبير السـابق قيمته تسـاوي 14 لأنه أولا يتم تنفيذ معامل الضرب على 3 و 4 الذي نتيجته 12 ومنه الجمع علـى 2 و 12 الذي ينتج 14 .

المعاملات في نفس المجموعة عند وجودها في التعبير فإنه يتم تنفيذها علـى حسـب ورودهـا . فـي المجموعـة الأولى بتم التنفيذ من اليمين إلى اليسار ، أما الثانية والثالثة فيتم تنفيـذها مـن اليسـار إلـى اليمـين . وكأمثلـة علـى هذا.

```
-+-10          (right to left)   →   (-(+(-10)))
10 * 2 / 5 * 4 (left to right)   →   (((10 * 2) / 5) * 4)
2 – 3 + 5      (left to right)   →   ((2 – 3) + 5)
```

المثال الأول لاحظ أن التنفيذ يتم من اليمين إلي اليسار وبالتالي يتم تنفيذ اشـارة النـاقص الأقرب إلـى العـدد أولا لترجع القيمة 10- ومنه اشارة الموجب لترجع 10- وأخيار السـالب لترجع 10 . المثال الثاني يتم تنفيذ أول عملية ضـرب على 10 و 2 لترجع 20 وبعدها القسـمة على 20 و 5 لترجع 4 وبعدها الضرب على 4 و 4 لترجع 16 . المثال الأخيـر يـتم تنفيذ أول عملية طرح لترجع 1- وبعدها الجمع على 1- و 5 لتكون النتيجة 4 . وأضع أيضا المثال التالي :

```
2 + 3 * 4 + 5
```

هنا يتم أولا حساب معامل الضرب لأن له أولوية أعلى مـن الجمـع . وبالتـالي التعبيـر يصـبح 5 + 12 + 2 الآن يـتم حساب الجمع من اليسار إلى اليمين بدأ من 2 و 12 ليصبح التعبير 5 + 14 وأخيرا تكون نتيجة التعبير 19 .

بالنسبة إلى عملية القسمة (/) فإن وظيفتها هي القسمة طبعا ولكن يجب الانتباه أنه عند قسمة عددين صحيحين integer فإن الجواب يكون عدد صحيح . أما في حالة كان أحد الحدين على الأقل عدد حقيقي float فإن الجواب يكون عدد حقيقي . وإليك بالأمثلة التالية :

13 / 5	تساوي	2
13.0 / 5	تساوي	2.6
13 / 5.0	تساوي	2.6

يعني في حالة القسمة الصحيحة فإن العدد الناتج هو القسم الصحيح من العدد ويزال مابعد الفاصلة . وكمثال على باقي القسمة لدينا :

13 % 5	تساوي	3
15 % 5	تساوي	0
-13 % 5	تساوي	-3
13 % -5	تسواي	3

في حالة أحد الحدين كان سالبا فإن اشارة باقي القسمة تكون مشابهة لأشارة المقسوم عليه كما في المثال السابق . وحتى نكون دقيقين أكثر فإن عملية باقي القسمة دائما تساوي إلى التعبير التالي :

a % b == a – (a / b) * b

حيث a و b هي أعداد صحيحة وقيمة b لاتساوي الصفر ، أي أن القسمة المستخدمة هي قسمة صحيحة .

- قمت بالاستفاضة في الشرح على هذه المعاملات السهلة لأنه في السي++ وكما سوف نرى لاحقا في قسم البرمجة غرضية التوجه Object Oriented Programming أنه يمكن اعادة برمجة العوامل من أجل اعطائها خواص جديدة .

2.9 التحويل بين الأنواع – Type conversions

في هذا القسم سوف نحاول الاجابة على السؤال التالي : ماذا يحدث إذا أردنا تنفيذ تعبير حسابي يتركب من حدود ذات أنواع مختلفة ؟

في البدية يجب أن أنوه على أن معظم العمليات الحسابية تتم على نوع int وذلك بسبب أن int دائما تكون بحجم مسجل المعالج الداخلي وبالتالي تكون العمليات الحسابية عليها أسرع مايمكن . علي سبيل المثال الأرقام الثابتة أو المحارف يتم تحويلها إلى int قبل القيام بالعمليات الحسابية عليها وذلك في حالة كانة ال int يمكن أن تحوي الرقم وإلا يتم استخدام unsigned int . طبعا هذا في حالة لم نقم باجبار الثابت أو الرقم على أن يكون long وذلك بذكر اللاحقة L بعد الرقم وعندها يجب أن نقوم بتحويلات أخرى على الأرقام أو المتحولات في التعبير .

القاعدة هنا بشكل عام أنه يتم تحويل الحدود من الأنواع الصغيرة إلى الأنواع الكبيرة . ويتم هذا على حسب الترتيب التالي :

- إذا كان أحد الحدين long double ، يتم تحويل الآخر إلى long double .
- إذا كان أحد الحدين double ، يتم تحويل الآخر إلى double .
- إذا كان أحد الحدين float ، يتم تحويل الآخر إلى float .
- نطبق عملية تحويل الint السابقة .
- إذا كان أحد الحدين unsigned long int ، يتم تحويل الآخر إلى unsigned long int . أما إذا كان أحد الحدود long int ، والآخر unsigned int وكانة ال long int يمكن أن تحوي جميع القيم الممكنة في unsigned int فإنه يتم تحويل الأخيرة إلى long int ؛ وإلا يتم تحويل الاثنين إلى unsigned long int .
- إذا كان أحد الحدين long int ، يتم تحويل الآخر إلى long int .
- إذا كان أحد الحدين unsigned int ، يتم تحويل الآخر إلى unsigned int .

العملية تبدو معقدة عند التعامل مع الأعداد الغير مؤشرة أو unsigned . على سبيل المثال اعتبر التصريح التالي:

```
unsigned int ui = 10;
         int si = -7;
```

واعتبر التعبير التالي : ui + si . بالاعتماد على القواعد السابقة فإن الحد الثاني سوف يحول إلى unsigned int . على فرض أن int هنا هي 16بت . فإن مجال القيم الممكن تمثيلها باستخدام unsigned int هو من 0 إلى 65535 . القيمة -7 بالنظام الست عشري تساوي 0xFFF9 وبالتالي إذا اعتبرنا القيمة السابقة كعدد غير مؤشر

(unsigned int) فإننـا نحصـل علـى القيمـة 65529 . وإذا جمعناهـا مـع القيمـة 10 نحصـل علـى 65539 أو 0x10003 بالنظام السـت عشـري وبالتالي يكون الجواب 3 لأننا نتعامل مع أعداد 16بت .

في الأنظمة الحديثة نجد أن int هو 32بت وأيضا long هـو 32بـت . فعمليـة التحويـل تصـبح أسـهل . وأيضا تجنـب اسـتخدام الأعداد الغير مؤشـرة قدر المسـتطاع وكن حذرا في حالة تم اسـتخدامها .

2.10 معامل الالحاق – The assignment operator

معامل الالحاق يسـمح بالحاق قيمة تعبير ما إلى متحول في البرنامج . وأبسـط صيغة إلى معامل الالحاق هي :

```
variable = expression;
```

عند تنفيذ عملية الالحاق فإنه يتم أولا حسـاب قيمة التعبير وبعدها يتم اعطـاء المتحـول القيمـة الجديـدة للتعبيـر . أيضا يتم تحويل نوع القيمة حتى توافق نوع المتحول . إن معامل الالحاق مثله مثل باقي المعـاملات لـه أولويـة وترتيـب في الحسـاب ، طبعا أولويته هي أصغر مايمكن ويتم حسـابه من اليمين إلى اليسار كما فـي معامـل الاشـارة السـالبة والموجبة . أوليته صغرى هذا يضمن تنفيذ التعبير قبل عملية الالحاق .

وكما أن التعبير a + b يرجع قيمة جديدة فأيضا التعبير a = b يرجع قيمة جديـدة وذلـك بعـد حسـاب قيمـة التعبيـر واسـنادها إلى المتحول ، هذه القيمة التي يرجعها تعبير الالحاق تسـاوي قيمة المتحول بعد الاسـناد . هـذا مفيـد جـدا في بناء سـلسـلة الاسـنادات مثل المثال التالي :

```
variable1 = variable2 = variable3 = … variablek = expression;
```

كما ذكرنا فإن تنفيذ معامل الالحاق يتم من اليمين إلى اليسار فإنه أولا يتم اسـناد قيمة التعبيـر expression إلـى المتحول variablek وبعدها قيمة هذا المتحـول تسـند إلـى variable(k-1) وهكـذا حتـى يـتم اسـنادها إلـى variable2 ومنه إلى variable1. ومنه فإن التعبير التالي :

```
X = y = z = p + q;
```

يتم تفسـيره على الشـكل التالي :

```
x = (y = (z = p + q));
```

كما ذكرنا فإن عملية التحويل تتم عبر معامل الالحاق فمثلا إذا أردنا الحاق ماهو من نوع int إلـى double فسـوف يتم تحويل ال int إلى ال double أما العكس أي من double إلى int فطبعا هـذه العمليـة نسـميها عمليـة مـدمرة لأن الرقم سـوف يقرب إلى أقرب عدد صحيح وقد يفقد من قيمته . على سـبيل المثال انظر ماذا سـوف يحـدث فـي التعـابير التالي :

int = int	تتم من غير تحويل
float = double	قص القيمة مع التقريب
int = long	قص القيمة إذا كانت int هي 16بت
char = int	قص القيمة دوما .

أيضا يجب الحرص عندما يتم اسـتخدام سـلسـلة الالحاقات مع أنواع مختلفة من المتحولات . إذا كانـت iii هـي مـن نوع int فإن التعبير التالي :

```
iii = 12.34;
```

يضع ويرجع القيمة 12 وبالتالي ففي التعبير التالي حيث fff هي من نوع float .

```
fff = iii = 12.34;
```

فإن القيمة 12 من نوع int سـوف تحول إلى float أي القيمة 12.0 سـوف يتم اسـنادها إلى المتحول fff .

2.11 معاملات الالحاق المركبة – The compound assignment operator

إن عملية الالحاق من الشكل

```
count = count + 2
```

تتكرر بشكل كبير في برامجنا وهي تعني أنه يتم اضافة مامقداره 2 إلى المتحول count وبعدها وضع القيمة الجديدة في المتحول count . أي بكلمة أخرى يتم اضافة اثنان إلى المتحول count . مثل الالحاق السابق يمكن كتابته في السي++ بالصيغة التالي :

```
count += 2
```

في الواقع فإن يمكن تطبيق نفس الأمر على باقي المعاملات الحسابية الأربعة وبالتالي يمكن أن نكتب مايلي :

```
count += 2
count -= 1
power *= 2.71828
divisor /= 10.0
remainder %= 10
```

في كل الحالات السابقة الحدين على طرفين المعامل يمكن أن يكونا من أي نوع ماعدا في الحالة الأخيرة حيث كما نعرف فإن معامل باقي القسمة يحتاج إلى نوع صحيح على طرفيه . أيضا فإن الأولية وطريقة التنفيذ هي تماما مثا معامل الالحاق الذي شرحناه في الفقرة السابقة .

أيضا يجب أن ننتبه إلى أن :

variable op= expression

تفسر كالتالي :

Variable = variable op (expression)

وبالتالي فإن المثال التالي: sum /= 3 + 7 يتم حسابه كالتالي : sum = sum / (3 + 7) .

أيضا يمكن استخدام سلسلة الالحاقات هنا أيضا ولكن مرة أخرى مع الحذر في بعض الحالات لاحظ المثال التالي:

```
fff = iii *= fff
```

إذا كانت fff من نوع float وتحتوي على القيمة 1.234 والمتحول iii من نوع int ويحتوي على القيمة 12 فإن التعبير السابق يتم حسابه كالتالي :

```
fff = (iii = iii * fff)
```

وبالتالي iii سوف يتم اسناد القيمة (12 * 1.234 = 14.808) 14 بعد ازالة القسم بعد الفاصلة. وبعدها يتم اسناد القيمة 14.0 للمتحول fff .

2.12 معاملا الزيادة والنقصان – The increment and decrement operators

في القسم السابق شرحنا معامل الاسناد المركب . ويمكن أن نستخدم معامل الاسناد المركب كالتالي من أجل اضافة القيمة 1 إلى المتحول :

```
x += 1
```

في الحقيقة فإن الزيادة بمقدار واحد هي من العمليات التي تمر علينا كثرة في برامجنا فلا يخلو برنامج منها . وأيضا هي من العمليات التي يتم تنفيذها بسرعة لأنها تكافئ إلى تعليمة واحده على الأغلب للمعالج . هذه العملية تسمى بمعامل الاضافة البعدية أو postincrement operator و معامل الاضافة القبلية أو preincrement operator . الثانية لها الصيغة :

```
++ variable
```

أما التعبير ذو الاضافة البعدية فهو على الصيغة :

```
variable ++
```

في كلا الحالتين السابقتين فإنه يتم الزيادة بمقدار واحد إلى المتحول variable . وأيضا كلا الحالتين السابقتين فإنها يرجعا قيمة . في الأولى (preincrement) فإنه يتم ارجاع القيمة الجديدة بعد الزيادة . أما الثانية (postincrement) فإنه يتم ارجاع قيمة المتحول قبل الزيادة . لو اعتبرنا المثال التالي :

```
sum += ++count
```

فإن المتحول count سوف يزاد بقيمة 1 ومن ثم يتم استخدام القيمة الجديدة في التعبير أي يتم اضافتها مع sum وتخزين النتيجة في sum . ولو اعتبرنا المثال التالي :

```
sum += count++
```

يتم اضافة قيمة المتحول count أيضا بمقدار 1 ولكن يتم استخدام القيمة السابقة لـcount في التعبير . على سبيل المثال إذا كانت قيمة sum و count هي 10 و 20 على الترتيب فإن count سوف تصبح 21 ويتم جمع القيمة 20 (القديمة) مع 10 من أجل أن نحصل على 30 وتخزينها في sum .

في حالة استخدام معامل الانقاص فاستخدامه تماما مثل معامل الزيادة ولكن يتم هنا استخدام (--) عوضا عن (++) ويتم الانقاص بمقدار 1 .

```
-- variable
variable --
```

بالنسبة إلى أولوية وكيفية تنفيذ هذين المعاملين فسوف تجدهل في جدول في آخر الدرس يجمع جميع المعاملات التي تم شرحها في هذا الدرس .

2.13 معامل التحويل – The type cast operator

جميع المعاملات السابقة كانة عبارة عن رموز ، أما هنا فالأمر مختلف ، فالمعامل هنا ليس ثابت وهو عبارة عن كلمة . تحدثنا سابقا عن عملية التحويل الضمنية التي تتم في التعابير الحسابية وخلال معامل الاسناد . أما عملية التحويل التي نقوم بها نحن فتسمى cast . إن الطريقة القياسية في عملية تحويل نوع إلى آخر هو عن طريق استخدام اسم النوع الذي نريد التحويل إليه ومنه نذكر اسم التابع أو القيمة بين قوسين كالتالي :

```
double(date)
```

هنا إذا كان المتحول date من نوع int فيتم تحويله إلى double . إن معامل التحويل تماما مثل باقي المعاملات أي له أولوية وكيفية في التنفيذ وهذا ماسوف تراه في آخر الدرس .

الطريقة السابقة في كتابة نوع التحويل هي تشابه طريقة استدعاء التوابع كما سوف نرى في دروس لاحقة . أما الطريقة الأخرى في كتابة التحويل هي التالية :

```
(type-specifier) expression
```

أي يتم كتابة النوع داخل أقواس ثم يليها التعبير المراد تحويله . هذه الطريقة مأخوذة من لغة السي ومازال يمكن استخدامها هنا من أجل التوافقية . وكأمثلة على هذه لدينا :

```
(double) date
(char) x
(int) d1 + d2
(int) (d1 + d2)
```

في المثال قبل الأخير سوف يتم تحويل d1 وجمع القيمة الجديدة مع d2 . أنا في المثال الأخير فسوف يتم جمع d1 و d2 ومن ثم القيام بعملية التحويل . هذا بسبب أن معاملات التحويل لها أولوية أعلى من الجمع وهذا ما سوف تراه في جدول الأولويات في آخر الدرس .

2.14 معامل الفاصلة – The comma operator

آخر المعاملات التي سوف نشرحها في هذا الدرس هو معامل الفاصلة التي تجد لها استخدامات في شتى التطبيقات . سوف أقوم بشرح سريع عليها على أن يأتي مزيد من الشرح والتطبيقات عليها لاحقا في دروس أخرى . إن معامل الفاصلة يتكون من تعبيرين يفصل بينها فاصلة كالتالي :

expression1 , expression2

بين كل المعاملات السابقة فإن معامل الفاصلة لديه الأولوية الدنيا حتى أقل من معامل الاسناد ويتم تنفيذها من اليسار إلى اليمين . وبالتالي في المثال السابق فإن expression1 يتم حسابها بشكل كامل ومن ثم expression2 . إن قيمة التعبير السابق تعتبر هي قيمة التعبير الثاني أي expression2 . مثال على هذا :

s = (t = 2, t + 3)

أولا يتم تنفيذ التعبير t = 2 وبعدها التعبير t + 3 الذي تكون نتيجته 5 ونتيجة تعبير الفاصلة السابق يكون 5 وبالتالي يتم اسناد 5 إلى المتحول s . طبعا لاحظ أهمية الأقواس في المثال السابق وإلا لاحظ المثال التالي :

s = t = 2, t + 3

هنا يتم اسناد القيمة 2 إلى كلا المتحولين s و t ومن ثم يتم تنفيذ التعبير t+3 الذي تكون قيمته 5 ولكن هذه القيمة التي تكون قيمة معامل الفاصلة يتم تدميرها لأنها غير مستخدمة .

أولويات المعاملات المذكورة في هذا الدرس :

(الجدول التالي مأخوذ من MSDN)

Symbol	Name or Meaning	Associativity
	Highest Precedence	
++ --	Post-increment Post-decrement	Left to right
++ --	Pre-increment Pre-decrement	Right to left
- +	Unary minus Unary plus	Right to left
(type)	Type cast [for example, (float) i]	Right to left
* / %	Multiply Divide Remainder	Left to right
+ -	Add Subtract	Left to right
=	Assignment	Right to left
*=, /=, %=, +=, -=	Compound assignment	Right to left
,	Comma	Left to right
	Lowest Precedence	